# DOTween

## 注意事项：

using DG.Tweening

初始化：,如果有使用过一次初始化则需要在任何的tween之前调用该方法一次

DOTween.Init(bool recycleAllByDefault = false（对象池回收）, bool useSafeMode = true（安全模式，自动处理销毁破坏之类的事情）, LogBehaviour logBehaviour = LogBehaviour.ErrorsOnly（只记录）)

DOTween.Init();

新的动画前需要DOKILL上一个动画，保证安全播放

Time.timeScale = 0 会将动画也暂停，可以通过在创建的动画加上SetUpdate(true)来保证动画的运行

transform.DOMoveX(5f, 2).SetUpdate(true);

创建的动画返回值是Tween

## 常用API

**1. Move 移动过渡**

**物体移动到指定位置。(世界坐标)**

transform.DOMove(new Vector3(10f, 2f, 0), 2f, false);

transform.DOMove(Vector3 to, float duration, bool snapping)

参数说明：

Vector3 to 目标位置

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

**沿着X/Y/Z轴移动到指定位置。(世界坐标)**

transform.DOMoveX(10, 2f, false);

transform.DOMoveY(10, 2f, false);

transform.DOMoveZ(10, 2f, false);

transform.DOMoveX/DOMoveY/DOMoveZ(float to, float duration, bool snapping)

参数说明:

float to 要移动到的坐标值

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

**物体移动到指定位置。(自身坐标)**

transform.DOLocalMove(new Vector3(10f, 2f, 0), 10f, false);

transform.DOLocalMove(Vector3 to, float duration, bool snapping)

参数说明：

Vector3 to 要移动到的位置

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

**沿着X/Y/Z轴移动到指定位置。(自身坐标)**

transform.DOLocalMoveX(10, 2f, false);

transform.DOLocalMoveY(10, 2f, false);

transform.DOLocalMoveZ(10, 2f, false);

transform.DOLocalMoveX/DOLocalMoveY/DOLocalMoveZ(float to, float duration, bool snapping)

参数说明：

float to 要移动到的轴的坐标

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

**跳跃到指定位置。（世界坐标）**

transform.DOJump(new Vector3(5, 0, 0), 10, 3, 1f, false);

transform.DOJump(Vector3 endValue, float jumpPower, int numJumps, float duration, bool snapping)

参数说明：

Vector3 endValue 最终要跳跃到的位置

float jumpPower 跳跃的强度，决定跳跃的高度(当前位置Y加上该值)

int numJumps 跳跃的次数

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

跳跃到指定位置，自身坐标

transform.DOLocalJump(new Vector3(10, 0, 0), 10, 3, 2f, false);

transform.DOLocalJump(Vector3 endValue, float jumpPower, int numJumps, float duration, bool snapping)

参数说明：

Vector3 endValue 最终要跳跃到的位置

float jumpPower 跳跃的强度，决定跳跃的高度(当前位置Y加上该值)

int numJumps 跳跃的次数

float duration 动画持续时间

bool snapping 默认为false，设为true时平滑地将所有值变为整数，即每次移动整数值。

———————————————————————————————————————————

**旋转到指定的值（根据欧拉角），世界坐标**

transform.DORotate(new Vector3(0, 90, 0), 0.1f, RotateMode.Fast);

transform.DORotate(Vector3 to, float duration, RotateMode mode)

参数：

Vector3 to 旋转的目标值

float duration 旋转过渡时间

RotateMode mode

Fast （默认值）采用最短路径，即旋转不会超过360°

FastBeyond360 旋转超过360°

WorldAxisAdd 与transform.Rotate(new Vector3(20, 0, 0)，Space.World)作用类似，最终值始终被视为相对值

LocalAxisAdd 与transform.Rotate(new Vector3(20, 0, 0)，Space.Self)作用类似，最终值始终被视为相对值

**欧拉角**

章动量θ、旋进角（即进动角）ψ和自转角φ组成。

往前走100m后右转

也就是自己的坐标来进行选择，需要确定一个正方向

会导致万向节死锁

利用四元数代替

———————————————————————————————————————————

**旋转到指定的值（四元数），世界坐标**

Quaternion targetRotation = Quaternion.Euler(0, 90, 0);//表示绕 Y 轴旋转 90 度

transform.DORotateQuaternion(targetRotation, 1f);

transform.DORotateQuaternion(Quaternion to, float duration)

参数：

Quaternion to 要旋转到的目标值四元数

float duration 旋转的过渡时间

**四元数：数据的范围是-1 - 1，每次旋转需要两个四元数**

**例如Quaternion.AngleAxis(float angle, Vector3 axis)，它可以返回一个绕轴线axis旋转angle角度的四元数变换。我们可以一个Vector3和它进行左乘，就将得到旋转后的Vector3。在Unity里只需要用一个“ \* ”操作符就可以进行四元数对向量的变换操作。如果我们想要进行多个旋转变换，只需要左乘其他四元数变换即可。**[【Unity】旋转的尽头是使用四元数让物体旋转\_unity 四元数 旋转-CSDN博客](https://blog.csdn.net/leoysq/article/details/136737304)

———————————————————————————————————————————

**旋转到指定的值（根据欧拉角），自身坐标**

transform.DOLocalRotate(new Vector3(0, 90, 0), 0.5f, RotateMode.Fast);

transform.DOLocalRotate(Vector3 to, float duration, RotateMode mode)

参数：

Vector3 to 旋转的目标值

float duration 旋转的过渡时间

RotateMode mode

Fast （默认值）采用最短路径，即旋转不会超过360°

FastBeyond360 旋转超过360°

WorldAxisAdd 与transform.Rotate(new Vector3(20, 0, 0)，Space.World)作用类似，最终值始终被视为相对值

LocalAxisAdd 与transform.Rotate(new Vector3(20, 0, 0)，Space.Self)作用类似，最终值始终被视为相对值

**旋转到指定的值（四元数），自身坐标**

———————————————————————————————————————————

Quaternion targetRotation = Quaternion.Euler(0, 90, 0);//表示绕 Y 轴旋转 90 度

transform.DOLocalRotateQuaternion(targetRotation, 0.5f);

transform.DOLocalRotateQuaternion(Quaternion to, float duration)

参数：

Quaternion to 要旋转到的目标值四元数

float duration 旋转的过渡时间

———————————————————————————————————————————

**朝向目标方向，Look At看向谁，即旋转目标使其朝向指定的位置方向。**

transform.DOLookAt(new Vector3(0, 90, 0), 0.5f, AxisConstraint.None, Vector3.up);

transform.DOLookAt(Vector3 towards, float duration, AxisConstraint axisConstraint = AxisConstraint.None, Vector3 up = Vector3.up)

参数：

Vector3 towards 旋转目标值

float duration 旋转总用时

AxisConstraint axisConstraint **旋转最终轴约束，只旋转此轴。（默认值为AxisConstraint.None）**

Vector3 up 定义向上方向的矢量。（默认值为Vector3.up）

朝向目标方向，Look At看向谁，即旋转目标使其朝向指定的位置方向，每帧更新 lookAt 位置（与此相反，当补间开始时，只计算一次 lookAt 旋转）

transform.DODynamicLookAt(new Vector3(0, 90, 0), 0.5f, AxisConstraint.None, Vector3.up);

transform.DODynamicLookAt(Vector3 towards, float duration, AxisConstraint axisConstraint = AxisConstraint.None, Vector3 up = Vector3.up)

参数：

Vector3 towards 旋转目标值

float duration 旋转总用时

AxisConstraint axisConstraint 旋转最终轴约束，只旋转此轴（默认值为AxisConstraint.None）

Vector3 up 定义向上方向的矢量（默认值为Vector3.up）

———————————————————————————————————————————

**3. Scale 缩放过渡**

**将物体放大（缩小）到指定的倍数或大小**

transform.DOScale(new Vector3(1.5, 1.5, 1.5), 0.5f);

transform.DOScale(2f, 0.5f);

transform.DOScale(float/Vector3 to, float duration)

参数：

float/Vector3 to 浮点数为倍数，向量为指定大小

float duration 放大/缩小总消耗时间

———————————————————————————————————————————

**对物体的某一轴方向进行放大（缩小）**

transform.DOScaleX/DOScaleY/DOScaleZ(1.5f, 1.5f);

transform.DOScaleX/DOScaleY/DOScaleZ(float to, float duration)

参数：

float to 放大到的倍数

float duration 放大/缩小总消耗时间

———————————————————————————————————————————

**4. Punch 冲击**

**受到冲击后的回弹效果**

transform.DOPunchPosition();

transform.DOPunchPosition(Vector3 punch, float duration, int vibrato, float elasticity, bool snapping)

参数：

Vector3 punch 要被击打到的最远位置（相对值，相对于局部坐标）

float duration 总持续时间

int vibrato 物体振动频率

float elasticity 值一般在0到1之间，0表示起点到冲击方向的震荡，1表示为一个完整的震荡，可能会超过起点，个人感觉后者效果更好。

bool snapping 是否进行平滑插值

———————————————————————————————————————————

**受到冲击后旋转效果**

transform.DOPunchRotation()

transform.DOPunchRotation(Vector3 punch, float duration, int vibrato, float elasticity)

参数：

Vector3 punch 要被击打到的角度（相对值，相对于局部坐标）

float duration 总持续时间

int vibrato 物体旋转频率

float elasticity 值一般在0到1之间，0表示最初角度到最大角度的旋转，1表示为一个完整的旋转，可能会超过最远角度。

**实现一个弹性效果，反复弹，最终复原。**

transform.DOPunchScale()

transform.DOPunchScale(Vector3 punch, float duration, int vibrato, float elasticity)

参数

Vector3 punch 弹到的大小

float duration 总持续时间

int vibrato 物体放缩频率

float elasticity 值一般在0到1之间，0表示最初角度到目标大小的放缩，1会产生负值，出现警告。

———————————————————————————————————————————

**5. Text 文本动画**

如果文本框内之前没有文字，在2s内逐字显示文字。如果原先有文字，则逐字覆盖掉原先文字，显示新文字，就像这样。

GetComponent<Text>().DOText("今天天气真不错!", 2);

## 进阶API

**1.DOTween.To()**

Vector3 myvalue = new Vector3(0, 0, 0);

DOTween.To(() => myvalue, x => myvalue = x, new Vector3(10, 10, 10), 2);

要移动的初始值 目标值 时间

———————————————————————————————————————————

DOTween.To(getter, setter, endValue, duration);

参数说明

getter 一个函数，用于获取当前值。它应该返回一个 float、Vector3、Color 等类型的值，具体取决于你要动画化的属性。

setter 一个函数，用于设置目标值。它接受一个参数（目标值），并将其应用于对象的属性。

endValue 动画的结束值。即动画完成时，属性应该达到的值。

duration 动画的持续时间，单位为秒。

———————————————————————————————————————————

示例 1: 动画化 float 值，targetValue 将在 2 秒内从 0 动画到 1。

float targetValue = 0f;

DOTween.To(() => targetValue, x => targetValue = x, 1f, 2f);

———————————————————————————————————————————

示例 2: 动画化 Vector3 值，物体的位置将在 3 秒内从当前的位置移动到 (0, 10, 0)。

Vector3 targetPosition = new Vector3(0, 0, 0);

Transform myTransform = gameObject.transform;

DOTween.To(() => myTransform.position, x => myTransform.position = x, new Vector3(0, 10, 0), 3f);

———————————————————————————————————————————

示例 3: 动画化 Color 值，物体的材质颜色将在 1 秒内变为红色。

Renderer myRenderer = GetComponent<Renderer>();

Color targetColor = Color.red;

DOTween.To(() => myRenderer.material.color, x => myRenderer.material.color = x, targetColor, 1f);

———————————————————————————————————————————

1. **From**

**从目标位置，移动到当前位置**

绝对位置，若当前坐标（1,0,0），即从5运动到1，**先从1闪现到5然后再在指定的时间内移动到1**

transform.DOMoveX(5, 1).From();

transform.DOMoveX(5, 1).From(false);

———————————————————————————————————————————

**相对位置，若当前坐标（1,0,0），即从6运动到1（6-1=5，相对位移5）**

transform.DOMoveX(5, 1).From(true);

1. **Set**

Tween tween = transform.DOMoveX(5, 2);

———————————————————————————————————————————

**.SetDelay(float delay):**

作用: 设置动画开始前的延迟时间（以秒为单位）。

示例:

tween.SetDelay(1f); // 延迟1秒后开始

———————————————————————————————————————————

**.SetEase(Ease easeType):**

作用: 设置动画的缓动类型（如线性、加速、减速等，类似设置PPT动画的出现效果）。

示例:

tween.SetEase(Ease.InOutQuad); // 使用 InOutQuad 缓动

不过问题不大，它有**明显的命名规则**，比如InSine、OutSine、InOut等，都是由“关键词”组成的，关键词就是下面这些：

**In指的是一种由慢到快的方式。**

**Out则指的是由快到慢。**

**Sine（正弦曲线）指的是比较平滑的过渡。**

**Quad则指的是会有更明显的快慢变化，比Quad速度变化更剧烈的，还有Cubic、Quart、Quint等。**

**Expo代表指数曲线，还有更多特殊的曲线，如有弹性的Elastic、先后退再前进的Back，以及Bounce（弹跳曲线）​**。

———————————————————————————————————————————

**.SetLoops(int loops, LoopType loopType):**

作用: 设置动画的循环次数和循环类型（如循环、反向循环等）。

示例:

tween.SetLoops(3, LoopType.Yoyo); // 循环3次，反向循环

———————————————————————————————————————————

**.SetAutoKill(bool autoKill):**

作用: 设置动画完成后是否自动销毁。

示例:

tween.SetAutoKill(false); // 动画完成后不自动销毁

———————————————————————————————————————————

**.SetId(object id):**

作用: 设置动画的 ID，以便后续查找和管理。

示例:

tween.SetId("myTween"); // 设置 ID 为 "myTween"

———————————————————————————————————————————

**.SetUpdate(bool isIndependent):**

作用: 设置动画是否独立于时间更新（用于在时间暂停时仍然更新）。

示例:

tween.SetUpdate(true); // 在时间暂停时仍然更新

———————————————————————————————————————————

1. **控制与回调**
   1. **开始暂停，结束**

**Tween tween = transform.DOMove(target.position, duration);**

**开始：DoMove,DOFade**

**暂停：Pause**

**恢复：Play**

**结束：Kill**

**重启：Restart**

**Tweener tweener = transform.DILocalMove(new Vector3(1, 1, 1), 1);**

**tweener.setAutoKill(false)**

**播放**

**tweener.Pause()**

**暂停**

**transform.Doplay()**

**向前播放动画**

**transform.DOPlayForward()**

**动画倒放**

**transform.DOPlayBackwards()**

**重新播放动画， 如果用了可视化编辑，需取消AutoKill （unity 界面配置大于代码配置）**

* 1. **回调：略**
  2. **事件：略**
  3. **注意事项**

**Tween 和Tweener的区别**

**Tweener** 是一个通用的动画对象，表示一个动画的实例。它可以是任何类型的动画，比如位置、旋转、缩放、颜色等。是所有动画的基类，**它可以被用来控制动画的播放状态，设置回调和链式动画**

**Tweener** 是Tween的特化、专门用于一种特定类型的动画，通常用于数值动画（浮点型、整数等）。允许你在动画中设置开始值和结束值，它通常用于需要从一个数值变换到另一个数值的情况。

1. **Id: 等同于标签.SetId("moveTween")**
2. **序列动画**

**DOTween.Sequence() PPT的连续动画，顺序执行，组合动画，控制时间**

//创建一个序列动画

Sequence loopTween = DOTween.Sequence();

//添加第一个动画：移动到目标位置

loopTween.Append(transform.DOMove(target.position, 1f).SetEase(Ease.Linear));

//添加第二个动画：旋转

loopTween.Append(transform.DORotate(new Vector3(0, 180, 0), 1f).SetEase(Ease.OutBounce));

//添加第三个动画：缩放

loopTween.Append(transform.DOScale(new Vector3(2, 2, 2), 1f).SetEase(Ease.OutElastic));

//播放序列动画

loopTween.Play();

//暂停序列动画

loopTween.Pause();

//停止序列动画

loopTween.Kill();

//重播序列动画

loopTween.Restart();

// 添加一个回调，在序列完成时执行

loopTween.OnComplete(() => Debug.Log("序列动画完成"));

**Sequence是Tween的一个动画系列功能的特化**

**其所特有的内容**

Sequence loopTween = DOTween.Sequence();

将一个新的动画添加到序列的末尾。

loopTween.Append(Tween tween);

在序列中添加一个回调函数（序列执行到该位置时会被调用）。

loopTween.AppendCallback(TweenCallback callback);

在序列中添加一个间隔（指定时间内不执行任何动画）。

loopTween.AppendInterval(float interval);

将一个新的动画添加到序列的开头。Prepend和Append刚好相反，一个添加到尾部(Append)一个添加到头部(Prepend)

loopTween.Prepend(Tween tween);

将一个回调函数添加到序列的开头。

loopTween.PrependCallback(TweenCallback callback);

在序列前面添加一个间隔（指定时间内不执行任何动画）。

loopTween.PrependInterval(float interval);

将一个新的动画添加到序列中，与当前动画同时播放。

loopTween.Join(Tween tween);

在指定时间插入一个新的动画。

loopTween.Insert(float time, Tween tween);

在特定时间点插入一个回调。

loopTween.InsertCallback(float atPosition, TweenCallback callback);

1. **官方文档** [DOTween - Documentation](https://dotween.demigiant.com/documentation.php)